An Availability-aware SFC placement Algorithm for Fat-Tree Data Centers

Ghada Moualla, Thierry Turletti, Damien Saucez

An Availability-aware SFC placement Algorithm for Fat-Tree Data Centers

Ghada Moualla, Thierry Turletti, Damien Saucez
Université Côte d'Azur, Inria, France

***Abstract*—Complex inter-connections of virtual functions form the so-called Service Function Chains (SFCs) deployed in the Cloud. Such service chains are used for critical services like e-health or autonomous transportation systems and thus require high availability. Respecting some availability level is hard in general, but it becomes even harder if the operator of the service is not aware of the physical infrastructure that will support the service, which is the case when SFCs are deployed in multi-tenant data centers. In this paper, we propose an algorithm to solve the placement of topology-oblivious SFC demands such that placed SFCs respect availability constraints imposed by the tenant. The algorithm leverages Fat-Tree properties to be computationally doable in an online manner. The simulation results show that it is able to satisfy as many demands as possible by spreading the load between the replicas and enhancing the network resources utilization.**

***Keywords*-SFC, Availability, Cloud, Data Center, Placement.**

I. Introduction

Network Function Virtualization (NFV) [1] virtualizes network functions and places them into commodity network hardware, such as a Data Center (DC). Since a single VNF cannot provide a full service, multiple VNFs are combined together in a specific order, called Service Function Chains (SFCs) [2]. SFCs determine the sequence of NFs that packets must follow and optimization techniques are used to map the SFCs in the network without overloading it and to provide availability guarantees.

Replication mechanisms have been proposed in the literature (e.g., [3], [4], [5]) to improve the required service availability based on VNF redundancy, which allow configurations in Active-Backup or Active-Active modes. However, some propositions [6] focus on replicating the SFCs in multi-tenant data centers where the tenant demands are oblivious to the actual physical infrastructure of the Data Center. Such an environment is particularly challenging as the demand is not known in advance and cannot be controlled. For the data center operator, it is therefore important to limit the number of replications to its minimum, yet respecting the level of service agreed with its tenants.

In this paper, we propose a placement algorithm for SFCs in Data Centers relying on Fat-Tree topologies. The algorithm is run by the network hypervisor and guarantees that Service Level Agreements (SLAs) with the tenants are respected, given the availability properties of the hardware deployed in the data center. Our proposition is based on an iterative linear program that solves the placement of SFCs in an online manner without prior knowledge on placement demand distribution. The algorithm is made computationally doable by leveraging symmetry properties of Fat-Tree topologies. Our evaluation on a very large simulated network topology (i.e., 27,648 servers and 2,880 switches) shows that the algorithm is fast enough for being used in production environments.

The rest of the paper is organized as follows. Section II presents the related work. Section III describes the problem statement. Section IV formulates the optimization model.. Section V proposes an availability-aware placement algorithm. Finally, Section VI evaluates the performance of our solution and Section VII concludes the paper.

II. Related Work

Multiple works tackle the problem of robust VM placement by deploying them on different physical nodes using specified availability constraints [7], [8], [9]. Zhang et al. [9] and Sampaio et al. [10] consider the MTBF of DC components to propose high availability placements of virtual functions in DCs. However, none of these works consider the benefits of using redundancy to ensure reliability. Rabbani et al. [11] solve the problem of availability-aware Virtual Data-Centers (VDC) embedding by taking into account components' failure rates when planning the number and the place of redundant virtual nodes but they do not consider the case of service chains.

In Herker et al. work [12], SFC requests are mapped to the physical network to build a primary chain, and backup chains are decided based on that primary chain. Engelmann et al. [13] propose to split service flows into multiple parallel smaller sub-flows sharing the load and providing only one backup flow for reliability guarantee. Our work follows the same principle as these two proposition but uses an active-active approach such that resources are not wasted for backup.

In our previous work [14], we proposed a deterministic solution for when SFCs are directly deployed by the DC owner and that requires to know in advance the minimum

number of replicas. In this paper, we propose a stochastic approach for the case where SFCs are requested by tenants oblivious to the physical DC network and that only have to provide the SFC they want to place and the required availability.

III. Problem Statement

This section defines the problem of placing SFCs in Data Centers under availability constraints.

Without any loss of generality, and inspired by works ([12], [15]), we only consider server and switch failures and ignore link failures. We also consider that all equipments of a same type have the same level of availability. More details are provided in Section VI-A.

A. Detailed description of the problem

This work develops an availability-oriented algorithm for resilient placement of VNF service chains in Fat-Tree based DCs where component failures are common [12].

The Fat-Tree topology is modeled as a graph where the vertices represent switching nodes and servers, while the edges represent the network links between them. Furthermore, SFC provides a chain of network functions with a traffic flow that need to traverse them in a specific order. We only consider acyclic SFCs. As we are in a multi-tenant scenario, functions are deployed independently and cannot be aggregated (i.e., function instances are not shared between SFC instances or tenants).

Each function is considered as a single point of failure. Thus, to guarantee the availability of a chain we use *scaled replicas*: we replicate the chain multiple times and equally spread the load between the replicas.

Upon independent failures, the total ***availability*** for the whole placed SFC replicas will be computed using Eq. (1) (availability for parallel systems).

$$availability = 1 - \prod_{i \in n\_replicas} (1 - ava_{sc_i}), \quad (1)$$

where ava_{sc_i} is the availability of replica i of service chain sc and $n\_replicas$ is the number of scaled replicas for this service chain. The availability of each service chain replica ava_{sc_i} is defined by

$$ava_{sc_i} = \prod_{f \in F} A_f, \forall i \in n\_replicas \quad (2)$$

where A_f is the availability of a service chain function f, which corresponds to the availability of the physical node that hosts this function ([6], [4]).

In Fat-Tree network topologies (Fig. 1), the availability of a system composed of multiple functions depends on its placement in the topology. For example, the availability of a SFC for the three scenarios for SFCs placement presented in Fig. 1 is calculated as follows: (i) `Scenario 1`, where service chain (`SC1`) is mapped to only one physical host, the availability of `SC1` is equal to the availability of its host node (h_1) while (ii) in `Scenario 2`, `SC2` is placed on 2 different hosts under the same ToR switch, so the availability of this service is equal to the availability of all participating host nodes in addition to the availability of their parent ToR switch as follows:

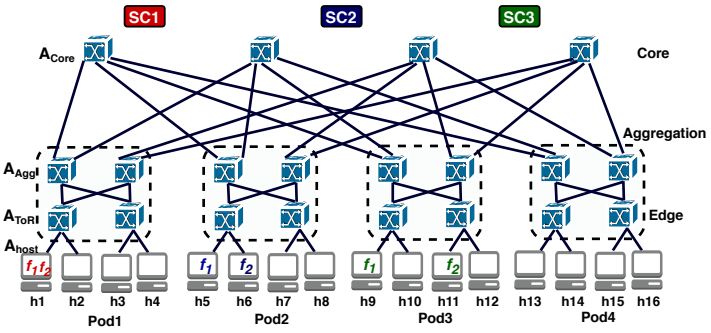

Figure 1. Fat-Tree topology example.

$$A_{SC2} = A_{h5}.A_{h6}.A_{ToR}. \quad (3)$$

However, in `Scenario3` where the chain `SC3` is placed on different hosts connected to different ToR switches within the same pod, the availability is given by:

$$A_{SC3} = A_{h9}.A_{h11}.(A_{ToR})^2 . \left[1 - (1 - A_{Agg})^2\right]. \quad (4)$$

In the general case of a k-ary Fat-Tree, each host is connected to one ToR switch (edge switch) which is directly connected to $k/2$ switch in aggregation layer and each switch in the aggregation layer is then connected to $k/2$ switches in the core layer. The generalized availability equation becomes:

$$A_{SC3} = (A_{host})^{n\_h}.(A_{ToR})^{n\_ToRs}. \left[1 - (1 - A_{Agg})^{k/2}\right], \quad (5)$$

where n_h refers to the total number of used hosts and n_ToRs refers to the total number of ToRs involved in the placement.

IV. Model Description and Formalization

A. Model Variables

We present here the variables used in our model formulation to place one particular service chain:

- $p \in P$: pod Id, $t \in ToRs$: ToR Id, $h \in Hosts$: host Id and F is a sequence of NFs id f of the SFC.
- π_p: Binary variable, equals to 1 if the pod p is used, and 0 otherwise.
- τ_α: Availability for the mapped service chain.
- $R \in [0, 1]$: SFC requested availability in the SLA.

- $\vartheta_{p,t,h,f}$: Binary variable, equals to 1 if the function f is mapped to a specific physical host identified by its pod p, its ToR t, and its Id h, and 0 otherwise.
- $\xi_{h,p,t}$: Binary variable, equals to 1 if the host h under the ToR t of the pod p is used, and 0 otherwise.
- $\rho_{t,p}$: Binary variable, equals to 1 if the ToR t on the pod p is used, and 0 otherwise.
- $\delta_{f,p}$: A binary variable that equals 1 if the function f is mapped to the pod p, and 0 otherwise.
- ε_p: Total number of used hosts under the pod p.
- σ_p: Total number of used ToRs under the pod p.
- C_f: Required CPU resources for function f.
- C_h: Total available CPU resources on host h.
- A_h, A_s: Availability of host h and switch node s, respectively.

B. Model Formulation

The optimization objective is to minimize the number of scaled replicas (i.e. number of used pods as each replica is placed in a different pod). This translates into:

$$Obj : Min \sum_{\mathbf{p} \in \mathbf{P}} \pi_{\mathbf{p}}. \tag{6}$$

Subject to the following constraints:

$$\tau_\alpha >= R \tag{7}$$

$$\forall_{h \in H, p \in P, t \in T} : \xi_{h,p,t} = 1 \text{ if} \sum_{f \in F} \vartheta_{p,t,h,f} \geq 1 \tag{8}$$

$$\forall_{t \in T, p \in P} : \rho_{t,p} = 1 \text{ if} \sum_{h \in H} \sum_{f \in F} \vartheta_{p,t,h,f} \geq 1 \tag{9}$$

$$\forall_{f \in F, p \in P} : \delta_{f,p} = 1 \text{ if} \sum_{t \in T} \sum_{h \in H} \vartheta_{p,t,h,f} \geq 1 \tag{10}$$

$$\forall_{p \in P} : \pi_p = 1 \text{ if} \sum_{t \in T} \sum_{h \in H} \sum_{f \in F} \vartheta_{p,t,h,f} \geq 1 \tag{11}$$

$$\forall_{p \in P},\ \forall_{f \in F}\ \forall_{f' \in F, f' > f} : \delta_{f,p} \leq \delta_{f',p} \tag{12}$$

$$\forall_{f \in F}, \forall_{p \in P} : \sum_{t \in T} \sum_{h \in H} \vartheta_{p,t,h,f} \leq 1 \tag{13}$$

$$\forall_{p \in P} :\ \varepsilon_p = \sum_{t \in T} \sum_{h \in H} \xi_{h,p,t} \tag{14}$$

$$\forall_{p \in P} :\ \sigma_p = \sum_{t \in T} \rho_{t,p} \tag{15}$$

$$\forall_{p \in P} : \alpha_p = \begin{cases} 0, & \text{if } \varepsilon_p = 0. \\ A_h, & \text{if } \varepsilon_p = 1. \\ A_h^{\varepsilon_p}.A_s, & \text{if } \varepsilon_p > 1 \text{ and } \sigma_p = 1. \\ A_h^{\varepsilon_p}.A_s^{\sigma_p}.(1-(1-A_s)^{\frac{k}{2}}), & \text{if } \sigma_p > 1. \end{cases} \tag{16}$$

$$\tau_\alpha = 1 - \prod_{p \in P} (1 - \alpha_p) \tag{17}$$

$$\forall_{h \in H} \sum_{p \in P} \sum_{t \in T} \sum_{f \in F} \vartheta_{p,t,h,f}\ C_f \leq C_h. \tag{18}$$

Constraint (7) ensures that the placement offers an availability at least as high as the one required in the SLA. Constraint (8) (resp. (9)) marks that the host (resp. ToR) is used if at least one NF is deployed on it (resp. on a host connected to it). Similarly, Constraint (11) indicates whether or not a specific pod is used. Constraint (10) indicates that a function f is placed on a specific pod p.

Constraint (12) ensures that if one function of a scaled service replica is placed in one pod p then all other functions of this replica are placed in this same pod. Constraint (13) ensures that two replicas of the same function are never placed in the same pod.

Equation (14) computes the total number of used hosts under each pod p, while Equation (15) counts the total number of used ToRs under each pod p in order to use them in Equation (16) to compute the availability of each scaled SFC replica. The latter takes one of four possible values based on the values we get from Equations (14) and (15). Finally, Constraint (18) ensures that the functions placed on a physical host node cannot use more CPU resources than its host resource capacity.

Constraints (16) and (17) are non-linear; we show how to linearize them in Appendix A. In our evaluation section, we used the linearized version of the model.

V. SFC Placement Algorithm

Directly solving the model of Sec. IV for a large DC topology is impractical. Instead we apply the model on a (small) subset of the topology, more precisely, only in one fault domain (pod), for the new model: equations (7, 17) will be removed, and the new objective is to maximize the placement availability over that fault domain :

$$Obj : Max(\alpha_p). \tag{19}$$

Our algorithm is called each time a request to install an SFC is received. Specifically, for a *required availability* R, the algorithm determines how many scaled replicas to create for that SFC and where to deploy them; taking into account the availability of network elements (servers and

switches) without impairing the availability guarantees of the chains already deployed. To guarantee the isolation between scaled replicas, each replica of a chain is deployed in a different fault domain.

Algorithm 1 presents the pseudo-code of our algorithm where `scale_down(`C`, `n`)` is a function that computes the scaled replica scheme, i.e., an annotated graph representing the scaled down chain, for a chain C if it is equally distributed over n scaled replicas and where `solve_placement(`S`, `G`, `n`)` solves the problem of placing n replicas S on the network topology G. The solution of a placement is a set of mappings associating replica functions and the compute nodes on which they have to be deployed. The solution is empty if no placement can be found.

Our algorithm starts with one replica of a service request and first checks that no function is requesting more resources than what the pod can offer.

In the case it is not possible to find a placement with one replica, the algorithm scales down the chain S by adding one more replica and tries to find a placement for each one of these replicas in different fault domains. Otherwise, the algorithm tries to find a placement for it under one fault domain of the network (the fault domain is chosen randomly to spread the load over the entire DC) using `solve_placement(`S`, `G`, `n`)` function; if no placement is found in the current fault domain, we check the another fault domain, otherwise we compute the total availability for the current placement.

This strategy continues until a termination condition is met: (i) if the requested availability is reached then the service can be deployed with (`deploy` (*placement)*); (ii) if the maximum acceptable time for finding a placement is reached then no solution is found; (iii) if the number of created scaled replicas reached the maximum number of replicas (i.e., maximum number of fault domains), then no solution is found. The `compute_ava` function computes the availability of a chain placement according to Sec. III.

Algorithm 1: Availability-aware placement

Input: Physical network Graph: G
chain $\in$ Chains
Scaled chain replica graph: *replica_scheme*
Required availability: R

T = *max_time*; $n = 1$; *tot_ava* = 0; *tot_time* = 0;
placement = ϕ; *replica_scheme* = *chain*
while *tot_ava*<*R* ***and*** *tot_time*<*T* ***and*** *n*<*max_n* **do**
  if *max_req* > *max_ava* **then**
    $n = n + 1$
    replica_scheme = `scale_down(`*chain, n*`)`
  else
    placement = `solve_placement(`
    replica_scheme, G, n`)`
    $n = n + 1$
    if ***not*** *placement* **then**
      replica_scheme = `scale_down(`*chain, n*`)`
    else
      tot_ava = `compute_ava(`*placement*`)`
  tot_time.update()

if *tot_ava* $\geq$ *R* **then**
  `deploy` (*placement*)

A. Scale down function

When multiple replicas are used, each one gets a fraction of the load and their individual resource requirements is lower than the one needed if there is only one chain instance. The resources depend on the availability of the other replicas and can be upper-bounded by:

$$\hat{R}_{f,n} = \left\lceil \frac{R_f}{n} + (n-1) \cdot \frac{C_f}{n} \cdot (1 - Ava_f) \right\rceil, \qquad (20)$$

where $\hat{R}_{f,n}$ is an upper-bound on the average amount of resources that would require a replica of a function f if it is replicated n times while R_f is the number of resources required by f in case it is not replicated at all, and Ava_f is the availability of the least available replica among the n replicas.

B. Solve placement function

The `solve_placement(`S`, `G`, `n`)` function considers two graphs: the DC topology graph G and the scale replica graph S. The purpose of this function is to project the scaled replica graph S on the topology graph G with respect to the physical and chain constraints.

For each fault domain, `solve_placement(`S`, `G`, `n`)` tries to find a solution for the linear problem defined earlier that aims at finding a placement for the scale replica graph in one fault domain while maximizing the availability of the replica placement.

VI. Evaluation

In the following we evaluate the Availability-aware placement algorithm introduced in the previous section.

A. Simulation Environment

We have implemented a discrete event simulator in Python interfaced with the Gurobi Optimizer 8.0 solver [16]. All simulations have been run on a Intel i7-4800MQ CPU at 2.70GHz and 32GB of RAM running

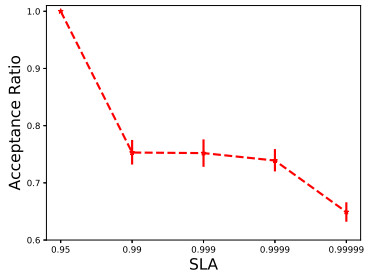

Figure 2. Comparing acceptance ratio for these different SLA values: 0.95, 0.99, 0.999, 0.9999, 0.99999.

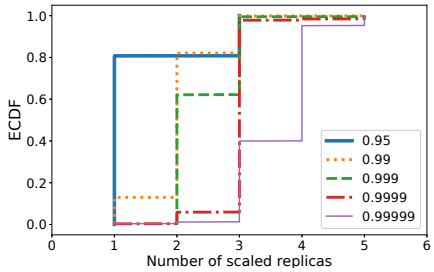

Figure 3. The ECDF for the number of created replicas with 5 different SLA with 48-Fat-Tree topology, $T_{IA} = 0.01$ and $S = 100$.

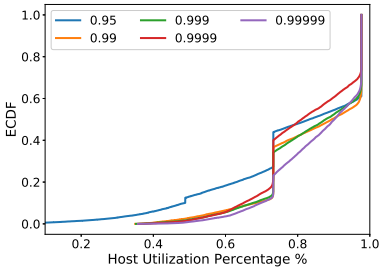

Figure 4. The ECDF for the host core utilization for different SLAs with 48-Fat-Tree topology, $T_{IA} = 0.01$ and $S = 100$.

GNU/Linux Fedora core 21.[1]

In the evaluation, requests to deploy SFCs are independent and follow an exponential distribution of mean inter-arrival time T_{IA} (measured in arbitrary time units). SFCs have a service time of S time units, i.e., the time the SFC remains in the system is randomly selected following an exponential distribution of mean S. If an SFC cannot be deployed in the network, it will be rejected. In total, our synthetic workload for the simulations contains 2,000 SFC request arrivals made of 20 random SFCs. As we are only interested in the steady state of the system, the servers are preloaded with service chains. All experiments presented here were repeated 5 times (5 different workloads of 2,000 SFC requests).

Furthermore, all SFCs are linear, i.e., they are formed of functions put in sequence between exactly one start point and one destination point. The number of NFs between the two endpoints is selected uniformly between 2 and 5, based on typical use cases of networks chains [17], and the requirements of each function in terms of cores is 1, 2, 4, or 8 inspired by the common Amazon EC2 instance types [18]. Simulations are performed on a 48-Fat-Tree topology with 48 pods having each 576 hosts for a total of 27,648 hosts. Every host has 4 cores. The availability of the physical devices in the Data Center are assigned accordingly to the statistical study of these works ([12], [15]), namely 0.99 for servers, 0.9999 for ToR and aggregation switches, 0.99999 for core switches, and 1.0 for links.

In the evaluation each SFC requires the same SLA even though the algorithm does not enforce it. In practice, placements must be computed in reasonable time so we limited the computation time to at most 6s per request, above that requests are rejected (the acceptable time for finding a placement must be at most of the same order of magnitude as the deployment of the VMs themselves to not impact the deployment time of a service).

[1]All the data and scripts used in this paper are available on https://team.inria.fr/diana/robstdc/.

B. Acceptance Ratio

The required availability level has an impact on the ability of a network to accept or not SFC requests. To study this impact, we consider the *acceptance ratio* defined as the number of accepted SFC requests over the total number of requests.

Figure 2 shows the evolution of the acceptance ratio w.r.t. the 5 different SLA levels. We can notice that the acceptance ratio decreases when the required availability level increases as each chain must reserve more resources than for lower availability levels as the physical topology is kept untouched. This can be explained by the fact that when increasing the required availability of a chain, it is necessary to replicate it further and then to consume more resources as at least one core is attributed to each function, replicated or not.

C. Level of Replication

To complete the acceptance ratio study, Figure 3 provides the Empirical Cumulative Distribution Function (ECDF) of the number of scaled replicas created for accepted SFCs for the different studied SLAs. It is clear that for the lowest required availability (0.95), 80% of SFCs were satisfied with exactly one replica as the availability of network elements are higher than this SLA level. However, when a SFC request needs more resources than the available resources in the network, it is split (20% of service were split for SLA=0.95) and, as the required availability increases, the required number of replicas are increased to satisfy the SFC SLA. Interestingly, as in practice the availability of the infrastructure is high we can observe than even for an aggressive SLA of 0.9999, 90% of SFC requests can be satisfied with no more than 3 replicas. Figure 3 shows that the number of replicas tops to 5 even though in theory it would be possible to observe

up to 48 replicas in a 48-Fat-Tree topology as there are 48 pods. We can explain this, as the computation time of our optimization is restricted to be less than 6 seconds.

Nevertheless, we can observe that a general increase of availability requirement increases the required number of replicas, which explains why the acceptance ratio decreases when the availability requirements increase.

D. Servers utilization

Figure 4 shows the ECDF of the server core utilization where the host utilization is the ratio between the total consumed CPU time and the total CPU time offered by the server. For example, for an experiment that lasts 2 units of time, if a server has 4 cores, the total CPU time offered by the server is 8. If during the experiment 3 functions are installed on the server and each function lasts 1.1 units of time and requires 2 cores, the total consumed CPU time is $3 \cdot 2 \cdot 1.1 = 6.6$, which means that the server is utilized at 82.5% of its capacity ($\frac{6.6}{8} = 82.5\%$).

In all scenarios, more than 40% of the servers are fully occupied. However as the required level increases, more servers CPU resources are used which explains why when the required availability increases, the overall load of the servers increases. When the required availability is as high as 0.99999, more than 80% of servers are more than 80% occupied. Nevertheless, even is highly loaded infrastructures, our algorithm can allocate resources in order to satisfy as much demands as possible.

VII. Conclusion

In this paper, we propose an online algorithm for SFC placement in data centers that leverages the Fat-Tree properties and respects the SFC availability constraints dictated by the tenant, taking into account the network components availability. The simulation results show that our algorithm is fast enough for being used in production environments and is able to satisfy as many demands as possible by spreading the load between the replicas while improving the network servers CPU utilization at the same time. For future work, we plan to extend our solution to consider other data center topologies, such as Leaf-and-Spine and BCube.

References

[1] ETSI, "Network Function Virtualisation (NFV); Architectural Framework," *NFV 001*, 2013.

[2] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," 2015.

[3] F. Machida, M. Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *IEEE NOMS*, 2010.

[4] J. Fan, C. Guan, K. Ren, and C. Qiao, "Guaranteeing availability for network function virtualization with geographic redundancy deployment," Tech. Rep., 2015.

[5] F. Carpio, W. Bziuk, and A. Jukan, "Replication of virtual network functions: Optimizing link utilization and resource costs," 2017.

[6] F. Carpio and A. Jukan, "Improving reliability of service function chains with combined vnf migrations and replications," *arXiv preprint arXiv:1711.08965*, 2017.

[7] M. Mihailescu, A. Rodriguez, and C. Amza, "Enhancing application robustness in Infrastructure-as-a-Service clouds," in *IEEE/IFIP DSN Conference*, pp. 146–151.

[8] D. Jayasinghe, C. Pu *et al.*, "Improving performance and availability of services hosted on IAAS clouds with structural constraint-aware virtual machine placement," in *IEEE SCC Conference*, 2011.

[9] Q. Zhang *et al.*, "Venice: Reliable virtual data center embedding in clouds," in *IEEE INFOCOM*, 2014.

[10] A. M. Sampaio *et al.*, "Towards high-available and energy-efficient virtual computing environments in the cloud," *Future Gener Comput Syst*, 2014.

[11] M. G. Rabbani *et al.*, "On achieving high survivability in virtualized data centers," *IEICE T COMMUN*, 2014.

[12] S. Herker *et al.*, "Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements," 2015.

[13] A. Engelmann *et al.*, "A reliability study of parallelized vnf chaining," *arXiv preprint arXiv:1711.08417*, 2017.

[14] G. Moualla, T. Turletti, and D. Saucez, "Robust placement of service chains in data center topologies," *https://team.inria.fr/diana/robstdc/*, 2018.

[15] P. Gill *et al.*, "Understanding network failures in data centers: measurement, analysis, and implications," 2011.

[16] G. Optimization, "Gurobi optimizer 5.0," *Gurobi: http://www.gurobi.com*, 2013.

[17] W. Liu *et al.*, "Service function chaining (SFC) general use cases," *IETF I-D draft-liu-sfc-use-cases-08*, 2014.

[18] "Amazon ec2 instance types." [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[19] B. Huo *et al.*, "Complete solution to a problem on the maximal energy of unicyclic bipartite graphs," *Linear Algebra and its Applications*, 2011.

Appendix

A. Linearization of the Nonlinear Constraints

Here we show how to linearize Constraints (16) and (17). For constraint (16) we introduce auxiliary binary

variables $\varepsilon_{p,i}$ for $i \in [0,n]$ where n is the maximum number of host nodes in one pod and $\sigma_{p,j}$ for $j \in [0,m]$ where m is the maximum number of ToRs in a pod. $\varepsilon_{p,i}$ refers to each possible number of used hosts under each pod, and $\sigma_{p,i}$ refers to each possible number of used ToRs under each pod. $\forall p \in P$:

$$\varepsilon_{p,i} = 1 \;\; if \; \varepsilon_p = i, \; else \; \varepsilon_{p,i} = 0, \tag{21}$$

$$\sigma_{p,j} = 1 \;\; if \; \sigma_p = j, \; else \; \sigma_{p,j} = 0. \tag{22}$$

Constraint (21) ensures that this variable equals to 1 when the total number of used host is equal to i else it is equal to 0. Then, Constraint (22) ensures that this variable equals to 1 when the total number of used ToR is equal to j else it is equal to 0.

$$\sum_{i=0}^{n} \varepsilon_{p,i} = 1 \tag{23}$$

$$\sum_{j=0}^{m} \sigma_{p,j} = 1 \tag{24}$$

Constraints (23, 25) and Constraints (24, 26) ensure that exactly one of $\varepsilon_{p,i}$ / $\sigma_{p,i}$ binary variables is equal to 1 respectively.

$$\varepsilon_p = \sum_{i=0}^{n} i \cdot \varepsilon_{p,i} \tag{25}$$

$$\sigma_p = \sum_{j=0}^{m} j \cdot \sigma_{p,j} \tag{26}$$

Now, we rewrite Constraint (16) as:

$$\begin{aligned} \forall_{p \in P} : \alpha_p = \varepsilon_{p,1}.A_h + \sigma_{p,1}.A_s.\Sigma_{i>1}^{n}\varepsilon_{p,i}.A_h^i + \\ \Sigma_{i>1}^{n}\varepsilon_{p,i}.A_h^i.\Sigma_{j>1}^{m}\sigma_{p,j}.A_s^j.(1-(1-A_s)^{\frac{k}{2}}). \end{aligned} \tag{27}$$

Regarding the other nonlinear constraint (i.e., Equation (17)), the following procedure will be considered to overcome the problem. Firstly, Equation (17) can be written as follows:

$$ln(1-\tau_\alpha) = \sum_{p \in P} ln(1-\alpha_p). \tag{28}$$

It is well known that for any real number $x > -1$, [19]:

$$ln(1+x) \leq x. \tag{29}$$

Inequality (29) is also held for any function of the form $F(x) = ax + b$, which is a tangent linear function. Therefore, this inequality can now be written as:

$$ln(1+x) \leq ax + b, \tag{30}$$

where the functions $F(x) = ax + b$ and $ln(1+x)$ have intersection at a certain point x_0. Note that a and b are constant, and they are calculated later. Based on Equation (30), we can write:

$$ln(1-\alpha_p) \leq -a_p\alpha_p + b_p \tag{31}$$

or

$$\sum_{p \in P} ln(1-\alpha_p) \leq \sum_{p \in P} (-a_p\alpha_p + b_p) \tag{32}$$

From Equation (28) and Inequality (29), we find that:

$$ln(1-\tau_\alpha) \leq \sum_{p \in P} (-a_p\alpha_p + b_p). \tag{33}$$

However, we need to make sure that $\tau_\alpha \geq R$, which translates into:

$$ln(1-\tau_\alpha) \leq ln(1-R) \tag{34}$$

Inequalities (33) and (34) do not guarantee whether $ln(1-R) \leq \sum_{p \in P} (-a_p\alpha_p + b_p)$ or the contrary.

But if we guarantee that the inequality $\sum_{p \in P} (-a_p\alpha_p + b_p) \leq ln(1-R)$ is met, Inequality (34) is then met, and consequently:

$$\sum_{p \in P} (-a_p\alpha_p + b_p) \leq ln(1-R). \tag{35}$$

As the function $F(x)$ is the tangent line, the constant a can be easily calculated by taking the differentiation of the function $ln(1+x)$ at a certain point x_0:

$$a = \frac{\mathrm{d}}{\mathrm{d}x}\bigg|_{x=x_0} F(x) = \frac{1}{1+x_0}. \tag{36}$$

The other constant b can be calculated by taking the value of the two functions at the point x_0, and thus:

$$ln(1+x_0) = ax_0 + b \tag{37}$$

or

$$b = -ax_0 + ln(1+x_0). \tag{38}$$

As our interest is to find whether $\tau_\alpha \geq R$, we need to intersect the tangent line with the logarithmic function at the point $x_0 = -R$, and thus $a_p = \frac{1}{1-R}$ and $b_p = \frac{R}{1-R} + ln(1-R)$.

Finally, in the formal model (IV-B), we replace Constraint (16) by the new Constraints (21, 22, 23, 24, 25, 26 and 27). While Constraints (17) and (7) are replaced by the new Constraint (35).